03 | 短 URL 生成器设计：百亿短 URL 怎样做到无冲突？

2022-02-21 李智慧

《李智慧 · 高并发架构实战课》 课程介绍 >

讲述：李智慧

时长 20:12 大小 18.50M

你好，我是李智慧。

从这节课开始，我们将结合具体的案例，来看看怎么设计高并发系统，你也可以学习具体的软件设计文档写法了。这个模块，我们先来看看，当高并发遇到海量数据处理时的架构。

在社交媒体上，人们经常需要分享一些 URL，但是有些 URL 可能会很长，比如：`https://time.geekbang.org/hybrid/pvip?utm_source=geektime-pc-discover-banner&utm_term=geektime-pc-discover-banner`

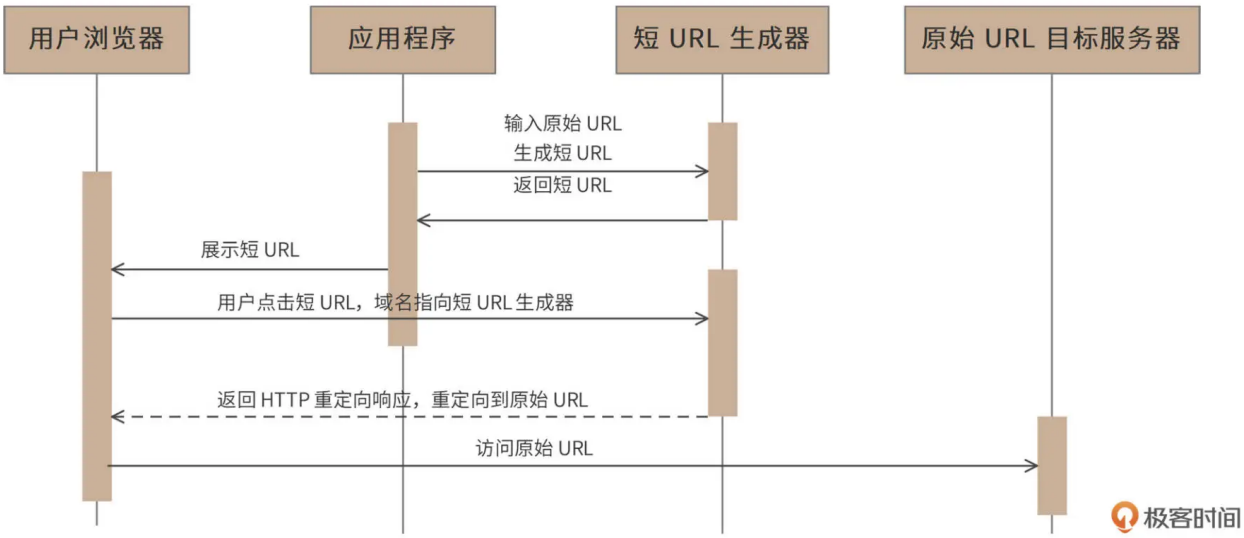
这样长的 URL 显然体验并不友好。我们期望分享的是一些更短、更易于阅读的短 URL，比如像 `http://1.cn/ScW4dt` 这样的。当用户点击这个短 URL 的时候，可以重定向访问

到原始的链接地址。为此我们将设计开发一个短 URL 生成器，产品名称是“Fuxi（伏羲）”。

我们预计 Fuxi 需要管理的短 URL 规模在百亿级别，并发吞吐量达到数万级别。这个量级的数据对应的存储方案是什么样的？用传统的关系数据库存储，还是有其他更简单的办法？此外，如何提升系统的并发处理能力呢？这些是我们今天要重点考虑的问题。

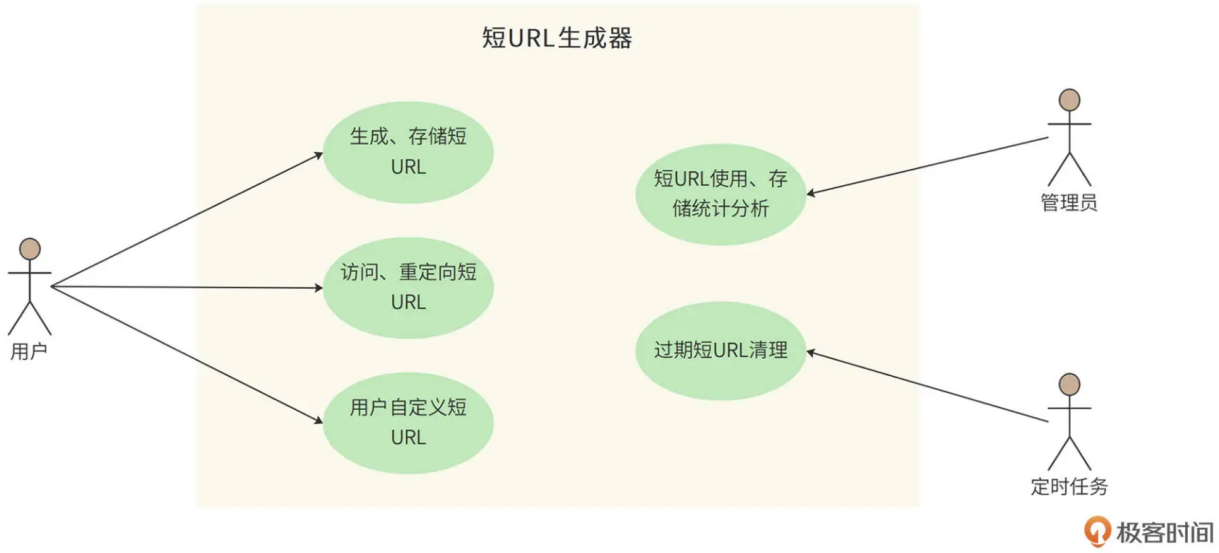
需求分析

短 URL 生成器，也称作短链接生成器，就是将一个比较长的 URL 生成一个比较短的 URL，当浏览器通过短 URL 生成器访问这个短 URL 的时候，重定向访问到原始的长 URL 目标服务器，访问时序图如下。



对于需要展示短 URL 的应用程序，由该应用调用短 URL 生成器生成短 URL，并将该短 URL 展示给用户，用户在浏览器中点击该短 URL 的时候，请求发送到短 URL 生成器（短 URL 生成器以 HTTP 服务器的方式对外提供服务，短 URL 域名指向短 URL 生成器），短 URL 生成器返回 HTTP 重定向响应，将用户请求重定向到最初的原始长 URL，浏览器访问长 URL 服务器，完成请求服务。

短 URL 生成器的用例图



1. 用户 client 程序可以使用短 URL 生成器 Fuxi 为每个长 URL 生成唯一的短 URL，并存储起来。
2. 用户可以访问这个短 URL，Fuxi 将请求重定向到原始长 URL。
3. 生成的短 URL 可以是 Fuxi 自动生成的，也可以是用户自定义的。用户可以指定一个长 URL 对应的短 URL 内容，只要这个短 URL 还没有被使用。
4. 管理员可以通过 web 后台检索、查看 Fuxi 的使用情况。
5. 短 URL 有有效期（2 年），后台定时任务会清理超过有效期的 URL，以节省存储资源，同时回收短 URL 地址链接资源。

性能指标估算

Fuxi 的**存储容量**和**并发量**估算如下。

预计每月新生成短 URL 5 亿条，短 URL 有效期 2 年，那么总 URL 数量 120 亿。

$$5亿 \times 12月 \times 2年 = 120亿$$

存储空间

每条短 URL 数据库记录大约 1KB，那么需要总存储空间 12TB（不含数据冗余备份）。

$$120亿 \times 1KB = 12TB$$

吞吐量

每条短 URL 平均读取次数 100 次，那么平均访问吞吐量（每秒访问次数）2 万。

$$（5亿 \times 100） \div （30 \times 24 \times 60 \times 60） \approx 20000$$

一般系统高峰期访问量是平均访问量的 2 倍，因此系统架构需要支持的吞吐能力应为 4 万。

网络带宽

短 URL 的重定向响应包含长 URL 地址内容，长 URL 地址大约 500B，HTTP 响应头其他内容大约 500B，所以每个响应 1KB，高峰期需要的响应网络带宽 40MB。

$$4万（每秒）次请求 \times 1KB = 40MB$$

Fuxi 的**短 URL 长度估算**如下。

短 URL 采用 Base64 编码，如果短 URL 长度是 7 个字符的话，大约可以编码 4 万亿个短 URL。

$$64^7 \approx 4万亿$$

如果短 URL 长度是 6 个字符的话，大约可以编码 680 亿个短 URL。

$$64^6 \approx 680亿$$

按我们前面评估，总 URL 数 120 亿，6 个字符的编码就可以满足需求。因此 Fuxi 的短 URL 编码长度 6 个字符，形如 http://l.cn/ScW4dt 。

非功能需求

1. 系统需要保持高可用，不因为服务器、数据库宕机而引起服务失效。
2. 系统需要保持高性能，服务端 80% 请求响应时间应小于 5ms，99% 请求响应时间小于 20ms，平均响应时间小于 10ms。

3. 短 URL 应该是不可猜测的，即不能猜测某个短 URL 是否存在，也不能猜测短 URL 可能对应的长 URL 地址内容。

概要设计

短 URL 生成器的设计核心就是短 URL 的生成，即长 URL 通过某种函数，计算得到一个 6 个字符的短 URL。短 URL 有几种不同的生成算法。

单项散列函数生成短 URL

通常的设计方案是，将长 URL 利用 MD5 或者 SHA256 等单项散列算法，进行 Hash 计算，得到 128bit 或者 256bit 的 Hash 值。然后对该 Hash 值进行 Base64 编码，得到 22 个或者 43 个 Base64 字符，再截取前面的 6 个字符，就得到短 URL 了，如图。

但是这样得到的短 URL，可能会发生 Hash 冲突，即不同的长 URL，计算得到的短 URL 是相同的（MD5 或者 SHA256 计算得到的 Hash 值几乎不会冲突，但是 Base64 编码后再截断的 6 个字符有可能会冲突）。所以在生成的时候，需要先校验该短 URL 是否已经映射为其他的长 URL，如果是，那么需要重新计算（换单向散列算法，或者换 Base64 编码截断位置）。重新计算得到的短 URL 依然可能冲突，需要再重新计算。

但是这样的冲突处理需要多次到存储中查找 URL，无法保证 Fuxi 的性能要求。

自增长短 URL

一种免冲突的算法是用自增长自然数来实现，即维持一个自增长的二进制自然数，然后将该自然数进行 Base64 编码即可得到一系列的短 URL。这样生成的的短 URL 必然唯一，而且还可以生成小于 6 个字符的短 URL，比如自然数 0 的 Base64 编码是字符“A”，就可以用 http://1.cn/A 作为短 URL。

但是这种算法将导致短 URL 是可猜测的，如果某个应用在某个时间段内生成了一批短 URL，那么这批短 URL 就会集中在一个自然数区间内。只要知道了其中一个短 URL，就可

以通过自增（以及自减）的方式请求访问其他 URL。Fuxi 的需求是不允许短 URL 可预测。

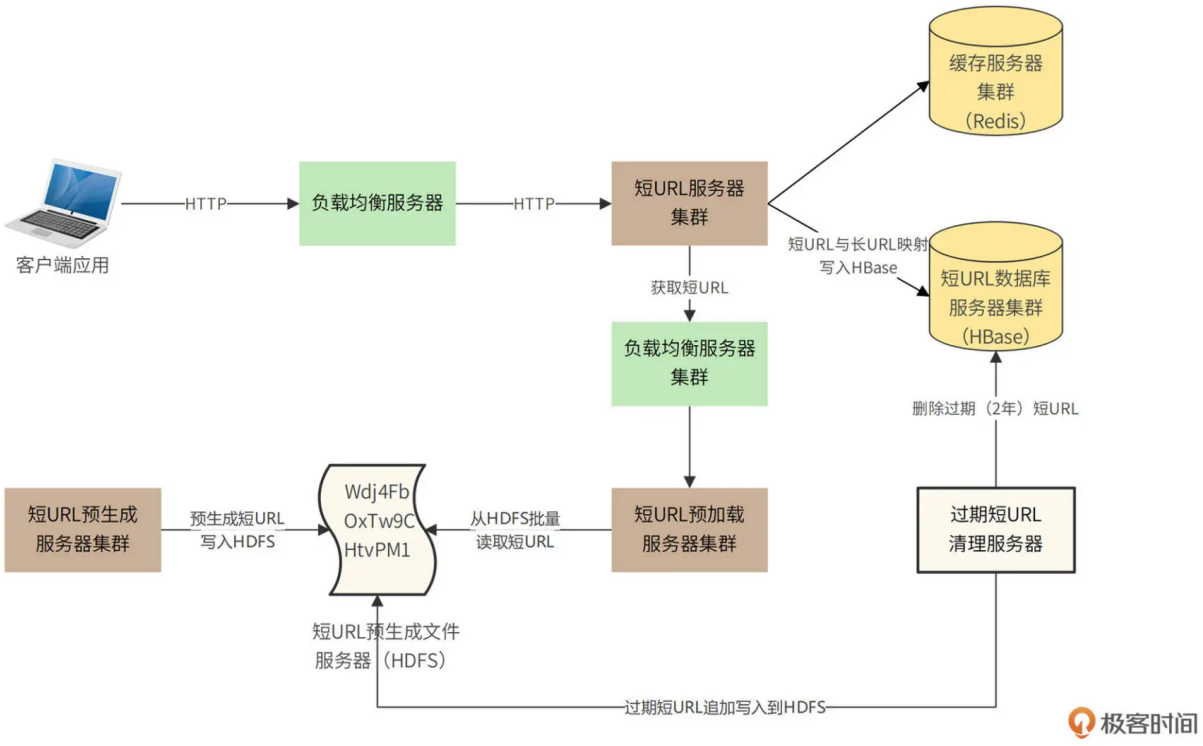
预生成短 URL

因此，Fuxi 采用预生成短 URL 的方案。即预先生成一批没有冲突的短 URL 字符串，当外部请求输入长 URL 需要生成短 URL 的时候，直接从预先生成好的短 URL 字符串池中获取一个即可。

预生成短 URL 的算法可以采用随机数来实现，6 个字符，每个字符都用随机数产生（用 0~63 的随机数产生一个 Base64 编码字符）。为了避免随机数产生的短 URL 冲突，需要在预生成的时候检查该 URL 是否已经存在（用布隆过滤器检查）。因为预生成短 URL 是离线的，所以这时不会有性能方面的问题。事实上，Fuxi 在上线之前就已经生成全部需要的 144 亿条短 URL 并存储在文件系统中（预估需要短 URL120 亿，Fuxi 预生成的时候进行了 20% 的冗余，即 144 亿。）

Fuxi 的整体部署模型

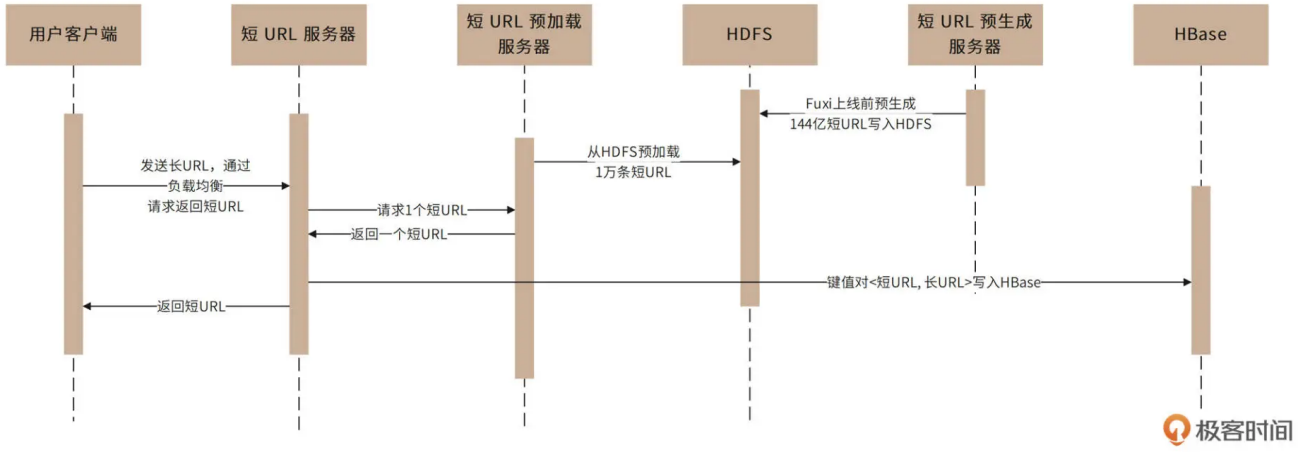
Fuxi 的业务逻辑比较简单，相对比较有挑战的就是**高并发的读请求如何处理、预生成的短 URL 如何存储以及访问**。高并发访问主要通过负载均衡与分布式缓存解决，而海量数据存储则通过 HDFS 以及 HBase 来完成。具体架构图如下。



系统调用可以分成两种情况，一种是用户请求生成短 URL 的过程；另一种是用户访问短 URL，通过 Fuxi 跳转到长 URL 的过程。

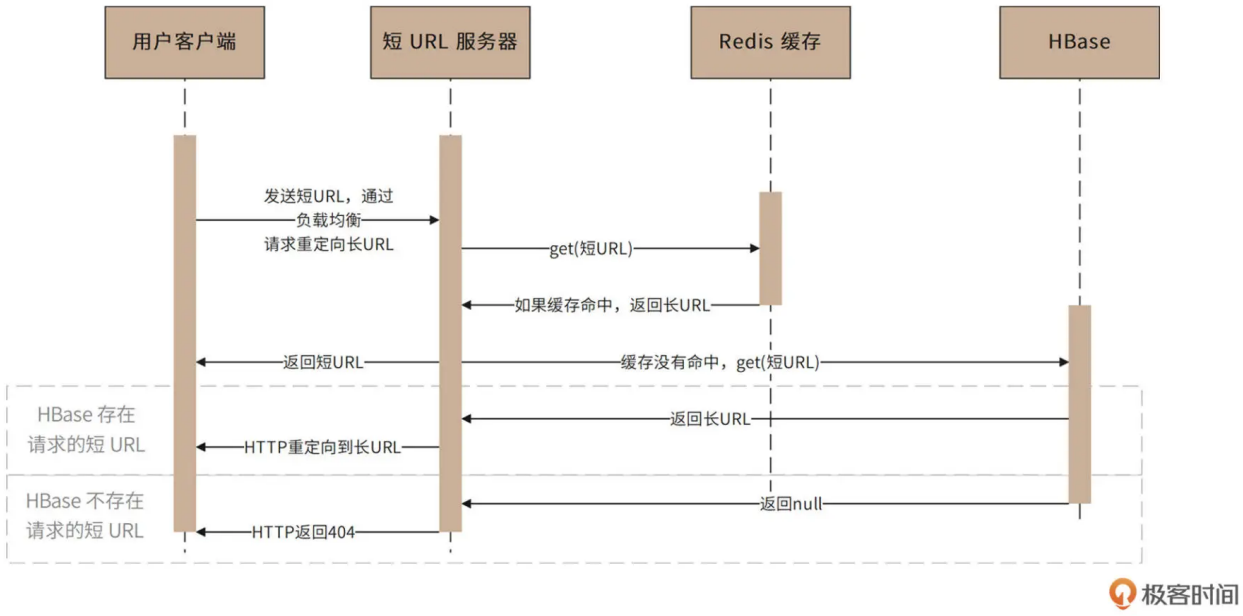
对于用户请求生成短 RUL 的过程，在短 URL 系统 Fuxi 上线前，已经通过随机数算法预生成 144 亿条短 URL 并将其存储在 HDFS 文件系统中。系统上线运行后，应用程序请求生成短 URL 的时候（即输入长 URL，请求返回短 URL），请求通过负载均衡服务器被发送到短 URL 服务器集群，短 URL 服务器再通过负载均衡服务器调用短 URL 预加载服务器集群。

短 URL 预加载服务器此前已经从短 URL 预生成文件服务器（HDFS）中加载了一批短 URL 存放在自己的内存中，这时，只需要从内存中返回一个短 URL 即可，同时将短 URL 与长 URL 的映射关系存储在 HBase 数据库中，时序图如下。



对于用户通过客户端请求访问短 URL 的过程（即输入短 URL，请求返回长 URL），请求通过负载均衡服务器发送到短 URL 服务器集群，短 URL 服务器首先到缓存服务器中查找是否有该短 URL，如果有，立即返回对应的长 URL，短 URL 生成服务器构造重定向响应返回给客户端应用。

如果缓存没有用户请求访问的短 URL，短 URL 服务器将访问 HBase 短 URL 数据库服务器集群。如果数据库中存在该短 URL，短 URL 服务器会将该短 URL 写入缓存服务器集群，并构造重定向响应返回给客户端应用。如果 HBase 中没有该短 URL，短 URL 服务器将构造 404 响应返回给客户端应用，时序图如下。



过期短 URL 清理服务器会每个月启动一次，将已经超过有效期（2 年）的 URL 数据删除，并将这些短 URL 追加写入到短 URL 预生成文件中。

为了保证系统**高可用**，Fuxi 的应用服务器、文件服务器、数据库服务器都采用集群部署方案，单个服务器故障不会影响 Fuxi 短 URL 的可用性。

对于 Fuxi 的**高性能**要求，80% 以上的访问请求将被设计为通过缓存返回。Redis 的缓存响应时间 1ms 左右，服务器端请求响应时间小于 3ms，满足 80% 请求小于 5ms 的性能目标。对于缓存没有命中的数据，通过 HBase 获取，HBase 平均响应时间 10ms，也可以满足设计目标中的性能指标。

对于 Redis 缓存**内存空间**估算，业界一般认为，超过 80% 请求集中在最近 6 天生成的短 URL 上，Fuxi 主要缓存最近六天生成的短 URL 即可。根据需求容量估计，最近 6 天生成的短 URL 数量约 1 亿条，因此需要 Redis 缓存服务器内存空间：$1亿 \times 1KB = 100GB$。

详细设计

详细设计关注重定向响应码、短 URL 预生成文件及加载、用户自定义短 URL 等几个关键设计点。

重定向响应码

满足短 URL 重定向要求的 HTTP 重定向响应码有 301 和 302 两种，其中 301 表示永久重定向，即浏览器一旦访问过该短 URL，就将重定向的原始长 URL 缓存在本地，此后不再请求短 URL 生成器，直接根据缓存在浏览器（HTTP 客户端）的长 URL 路径进行访问。

302 表示临时重定向，每次访问短 URL 都需要访问短 URL 生成器。

一般说来，使用 301 状态码可以降低 Fuxi 服务器的负载压力，但无法统计短 URL 的使用情况，而 Fuxi 的架构设计完全可以承受这些负载压力，因此 Fuxi 使用 302 状态码构造重定向响应。

短 URL 预生成文件及预加载

Fuxi 的短 URL 是在系统上线前全部预生成的，并存储在 HDFS 文件中。共 144 亿个短 URL，每个短 URL 6 个字符，文件大小 $144亿 \times 6B = 86.4GB$。

文件格式就是直接将 144 亿个短 URL 的 ASC 码无分割地存储在文件中，如下是存储了 3 个短 URL 的文件示例：

复制代码

```
Wdj4FbOxTw9CHtvPM1
```

所以如果短 URL 预加载服务器第一次启动的时候加载 1 万个短 URL，那么就从文件头读取 60K 数据，并标记当前文件偏移量 60K。下次再加载 1 万个短 URL 的时候，再从文件 60K 偏移位置继续读取 60K 数据即可。

因此，Fuxi 除了需要一个在 HDFS 记录预生成短 URL 的文件外，还需要一个记录偏移量的文件，记录偏移量的文件也存储在 HDFS 中。同时，由于预加载短 URL 服务器集群部署多台服务器，会出现多台服务器同时加载相同短 URL 的情况，所以还需要利用偏移量文件对多个服务器进行互斥操作，即**利用文件系统写操作锁的互斥性实现多服务器访问互斥**。

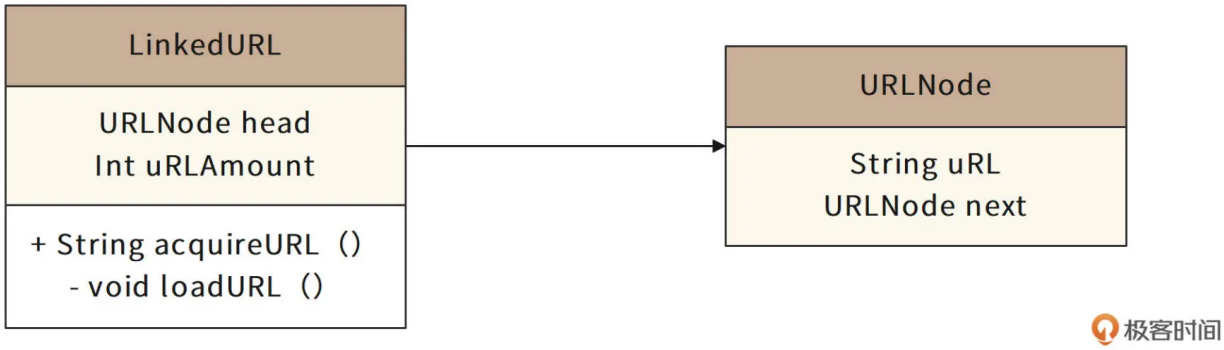
应用程序的文件访问流程应该是：写打开偏移量文件 -> 读偏移量 -> 读打开短 URL 文件 -> 从偏移量开始读取 60K 数据 -> 关闭短 URL 文件 -> 修改偏移量文件 -> 关闭偏移量文件。

由于写打开偏移量文件是一个互斥操作，所以第一个预加载短 URL 服务器写打开偏移量文件以后，其他预加载短 URL 服务器无法再写打开该文件，也就无法完成读 60K 短 URL 数据及修改偏移量的操作，这样就能保证这两个操作是并发安全的。

加载到预加载短 URL 服务器的 1 万个短 URL 会以链表的方式存储，每使用一个短 URL，链表头指针就向后移动一位，并设置前一个链表元素的 next 对象为 null。这样用过的短 URL 对象可以被垃圾回收。

当剩余链表长度不足 2000 的时候，触发一个异步线程，从文件中加载 1 万个新的短 URL，并链接到链表的尾部。

与之对应的 URL 链表类图如下。



URLNode：URL 链表元素类，成员变量 uRL 即短 URL 字符串，next 指向下一个链表元素。

LinkedURL：URL 链表主类，成员变量 head 指向链表头指针元素，uRLAmount 表示当前链表剩余元素个数。acquireURL() 方法从链表头指针指向的元素中取出短 URL 字符串，并执行 urlAmount-- 操作。当 urlAmount < 2000 的时候，调用私有方法 loadURL()，该方法调用一个线程从文件中加载 1 万个短 URL 并构造成链表添加到当前链表的尾部，并重置 uRLAmount。

用户自定义短 URL

Fuxi 允许用户自己定义短 URL，即在生成短 URL 的时候，由用户指定短 URL 的内容。为了避免预生成的短 URL 和用户指定的短 URL 冲突，Fuxi 限制用户自定义短 URL 的字符个数，不允许用户使用 6 个字符的自定义短 URL，且 URL 长度不得超过 20 个字符。

但是用户自定义短 URL 依然可能和其他用户自定义短 URL 冲突，所以 Fuxi 生成自定义短 URL 的时候需要到数据库中检查冲突，是否指定的 URL 已经被使用，如果发生冲突，要求用户重新指定。

URL Base64 编码

标准 Base64 编码表如下。

数值	字符	数值	字符	数值	字符	数值	字符
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

极客时间

其中“+”和“/”在 URL 中会被编码为“%2B”以及“%2F”，而“%”在写入数据库的时候又和 SQL 编码规则冲突，需要进行再编码，因此直接使用标准 Base64 编码进行短 URL 编码并不合适。URL 保留字符编码表如下。

!	#	$	&	'	(	)	*	+	,	/	:	;	=	?	@	[	]
%21	%23	%24	%26	%27	%28	%29	%2A	%2B	%2C	%2F	%3A	%3B	%3D	%3F	%40	%5B	%5D

所以，我们需要针对 URL 场景对 Base64 编码进行改造，使用 URL 保留字符表以外的字符对 Base64 编码表中的 62，63 进行编码：将“+”改为“-”，将“/”改为“_”，Fuxi 最终采用的 URL Base64 编码表如下。

数值	字符	数值	字符	数值	字符	数值	字符
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	-
15	P	31	f	47	v	63	_

极客时间

小结

我们开头提到，Fuxi 是一个高并发（2 万 QPS）、海量存储（144 亿条数据）、还需要 10ms 的高性能平均响应时间的系统。但是我们后面看到，Fuxi 的架构并不复杂。

这一方面是源于 Fuxi 的业务逻辑非常简单，只需要完成短 URL 与长 URL 的映射关系生成与获取就可以了。另一方面则是源于开源技术体系的成熟，比如一个 HDFS 集群可支持百万 TB 规模的数据存储，而我们需要的存储空间只有区区不到 100GB，都有点大材小用了。事实上，Fuxi 选择 HDFS 更多的考量是利用 HDFS 的高可用，HDFS 的自动备份策略为我们提供了**高可用的数据存储解决方案**。

同理，高并发也是如此，2 万 QPS 看起来不小，但实际上，由于业务逻辑简单，单个数据都很小，加上大部分请求数据可以通过 Redis 缓存获取，所以实际响应时间是非常短的，10ms 的平均响应时间使得 Fuxi 真正承受的并发压力只有 200。对于这样简单的业务逻辑以及 200 这样的并发压力，我们使用配置高一点的服务器的话，只需要一台短 URL 服务器其实就可以满足了。所以，我们在短 URL 服务器之前使用负载均衡服务器，这也是更多地为**高可用**服务。

思考题

用户每次请求生成短 URL 的时候，Fuxi 都会返回一个新生成的短 URL，也就意味着，如果用户重复提交同一个长 URL 请求生成短 URL，每次都会返回一个新的短 URL。你认为这将导致什么问题？对此，你有什么解决方案？

另外，小结里提到，2 万 QPS，10ms 平均响应时间，这种情况下，真正的并发量只有 200，这个 200 是如何得到的？

欢迎在评论区分享你的思考，或者提出对这个设计文档的评审意见，我们共同进步。

分享给需要的人，Ta订阅超级会员，你将得 50 元

Ta单独购买本课程，你将得 20 元

生成海报并分享

赞 6 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 02 | 高并发架构设计方法：面对高并发，怎么对症下药？

下一篇 04 | 网页爬虫设计：如何下载千亿级网页？

精选留言 (19)

写留言

Jialin 神

2022-02-21

问题一:如何解决相同的原始链接生成不同的短链接？
当要给一个原始网址生成短网址的时候，我们要先拿原始网址在数据库中查找，看数据库中是否已经存在相同的原始网址了。如果数据库中存在，那我们就取出对应的短网址，直接返回给用户。并给短网址和原始网址这两个字段，都添加索引。短网址上加索引是为了提高用户查询短网址对应的原始网页的速度，原始网址上加索引是为了加快通过原始网...

展开

作者回复: 1 索引的代价有点大的，我们再考虑下
2 是的

6

老码不识途
2022-02-21

再加一个架构的约束：只有2万块钱的预算 ^_^

作者回复: 大佬，请上座。

6

向东是大海
2022-02-21

短URL使用URL Base64 编码，其中的-和_字符显得有点突兀，特别是在开头或结尾是时。短URL使用Base62编码是否更好？

作者回复: 赞，很好的意见。

4

极客
2022-02-21

加载到预加载短 URL 服务器的 1 万个短 URL 会以链表的方式存储，每使用一个短 URL，链表头指针就向后移动一位，并设置前一个链表元素的 next 对象为 null。这样用过的短 URL 对象可以被垃圾回收。当剩余链表长度不足 2000 的时候，触发一个异步线程，从文件中加载 1 万个新的短 URL，并链接到链表的尾部
-----...
展开

作者回复: 预生成多20%的冗余，一般重启丢失一些是可以接受的，频繁重启可能系统有大麻烦了，需要另外思考了。

继续生成机制可以在系统运行期考虑，如果短URL消耗快于预期，可以增加再生成机制。架构不必一步到位。

共 2 条评论 >

2

2022-02-23

第一个问题：

重复生成同一个长URL，带来的问题主要是资源浪费。并且没有处理用户的重复请求。
可以考虑的解决方案是缓存，将每个长URL作为参数缓存起来。保存一段时间，只需要存几天即可。下次请求的时候，地址如果还是同样的长URL，并且命中缓存，那就把现有的...
展开

作者回复: right

1

Spark
2022-02-22

对自增长短 URL可以进行如下优化：
在设计的短链接的时候采用这样的方法：
根据自增主键id将前面补0（4位短链接补至7位，6位短链接补至10位，8位短链接补至14位）如主键：123，补0至0000123
倒转补完0之后的id，如3210000...
展开

作者回复: 很赞

2

jiangjing
2022-02-21

1s / 1ms = 100 每秒可处理100个请求
2w / 100 = 200 一秒内用一时刻只需要处理200请求

疑问：一般并发就是指qps / tps 吧？还是指这个计算出的200？

作者回复: 算法是对的，不过似乎应该是1s / 10ms =100

并发不是TPS，第二篇专栏有讲到"只有发起请求，在服务器正在处理这个请求的用户才是并发用户。"
后面讲性能架构专题的时候，会专门讲下这几个指标，TPS QPS这些叫吞吐量。

共 3 条评论 >

1

linuxcoder
2022-02-21

思考一：
如果用户重复提交同一个长 URL 请求生成短 URL，每次都会返回一个新的短 URL，这样会导致多个短URL对应一个长URL。如果需求只允许1对1，可以在返回生成的短URL之前去HBase查一下是否生成过了。如果允许N个短URL对应一个长URL，也可以从HBase查一下是否超过了N，如果没有，则返回生成的一个段URL。...
展开

作者回复: 1 多对一在功能上其实没问题的，都可以正确引用短URL，并正确跳转。HBase存储的key是短URL，如果用长URL遍历，性能太差了，无法接受。

2 对

1

来需求了，很忙
2022-02-24

为什么生成的短链不直接放到HBASE，被使用的短链去做update，这样不是简单点，也不用引入hdfs

作者回复: 生成的时候查HBase，如何索引未使用过的短链？似乎比HDFS麻烦

刘帅
2022-02-23

问题一：原始链接重复生成短url会造成存储资源浪费，计算性能浪费等问题，可以使用布隆过滤器对原始链接进行过滤，存在对应的短url直接返回即可
问题二：2Wqps*10ms=200并发度，即同一时刻只需处理200个并发用户的请求就可满足2Wqps的性能要求
展开

作者回复: √

A
2022-02-22

面试的时候一上来就是百万并发😠

作者回复: 人艰勿拆，那我们就互相吹吧~~

共 2 条评论 >

G小调

2022-02-22

有源码吗

李焕之

2022-02-22

李老师好，有两个疑问：1、文中说：每条短 URL 数据库记录大约 1KB。为什么会这么大呢？短+长串应该超不过200字节吧？还是考虑可能会有极端长的url？ 2、6位base64码680亿，预先生成了120亿，还是有很大概率可以使用短URL去撞到长URL呀,六分之一的概率，需要考虑防刷吗？

展开

作者回复: 1 长URL通常不止200B，同时还需要记录生成时间、请求者信息等数据，1KB其实是保守的。

2 文中提到，预生成的时候做冲突检查的，保证短URL唯一。

halweg

2022-02-21

并发数200，需要200个cpu核心吗？

作者回复: 不需要，并发200需要200个线程处理，线程可用共享CPU核心。

共 4 条评论 >

Tianium

2022-02-21

需求没有分析生成短url的规模和性能指标吧，一般生成短url的性能可以允许差点，如果不是，那问题一就不能简单用查hbase来解决了，而需要利用列入课程提到的bloom filter以及分布式缓存进一步加速了。

作者回复: 赞，你的意见很重要，生成短URL的资源消耗不可忽略的。
用布隆过滤器加速生成URL的冲突检查也是很好的方案。

共 2 条评论 >

Tim
2022-02-21

1）重复请求生成不同的短链没有问题，这就是老师预留20%多余短链的目的。如果去重的话，可以对长链采用布隆过滤判定是否已经占用
2）200/0.01=20000，反之。

作者回复: 1 我的建议是缓存一小时内的短链，把重复控制到1小时以上
2 对

共 5 条评论 >

linuxcoder
2022-02-21

请问老师，通过hdfs预加载方案，只有一台服务器能读取偏移量，然后读到10k个地址，放入内存，那负载均衡器如何知道哪个服务器有数据呢？需要把这个10k个短地址放入分布式cache吗？否则的话会有性能瓶颈吗？谢谢

作者回复: “只有一台服务器能读取偏移量”是为了控制并发，意思是“在同一个时间，只有一台服务器能读取偏移量”，事实上，一台服务器取完短URL，释放偏移量文件，另一个服务器就可以去取了，最后，所有服务器都有短URL。负载均衡可以请求任何服务器。

共 3 条评论 >

KK_TTN
2022-02-21

为什么 "短 URL 应该是不可猜测的" 是非功能需求?

作者回复: 你的疑问有道理，估计是作者偷懒吧，不想分开写~

共 3 条评论 >

shen
2022-02-21

2wqps是指每秒钟2w请求，现在10ms响应结果那么平均并发量20000*10/1000=200

作者回复: 对